# 类对象

类对象:代表类本身的对象，可以利用对象来获取类本身的相关的信息，比如这个类的属性有哪些，有哪些方法，有哪些构造方法，有哪些方法是公有的(public)和私有的(private)

# 获取类对象的方法

## 类名.class

## GetClass方法

## Class.forName()方法

### 给类名的时候最好直接给完整的类名和包名。

## 代码实例

|  |
| --- |
| package com.oracle.test;  public class Test {  /\*\*  \* @param args  \* 类对象：代表类本身的对象,可以利用这个类对象来获取类本身的相关的信息比如这个类的属性有哪些，有哪些方法，  \* 有哪些构造方法，有哪些方法是公有的(public)和私有的（private）  \*/  public static void main(String[] args) {  // 获取类对象的方式三种方式  // 1.类名称.class  Class clazz1 = Student.class;  // 2.向创建对象使用对象的方法getClass()获得类对象  Student s = new Student();  Class clazz2 = s.getClass();  // 3使用类全限路劲包名+类名  try {  Class clazz3 = Class.forName("com.oracle.test.Student");  } catch (ClassNotFoundException e) {  e.printStackTrace();  }  }  } |

# 反射包

## Filed类

### 获取和设置属性的值：

|  |
| --- |
| package com.oracle.test;  import java.lang.reflect.Field;  public class Test4 {  /\*\*  \* @param args  \* 类对象：代表类本身的对象,可以利用这个类对象来获取类本身的相关的信息比如这个类的属性有哪些，有哪些方法，  \* 有哪些构造方法，有哪些方法是公有的(public)和私有的（private）  \*/  public static void main(String[] args) {  // 使用反射的方式给对象的属性赋值  // 1.创建学生对象  Student stu = new Student();  // 原来的属性赋值方式  // stu.name = "zzz";  // 原来的获取属性值方式  // String nameValue = stu.name;  // 使用反射的方式给对象的属性赋值  Class c = stu.getClass();  try {  Field f = c.getField("name");  f.set(stu, "zzz");  // 使用反射的方式获取属性的值  String nameValue = (String) f.get(stu);  System.out.println("获取的值为:" + nameValue);  } catch (NoSuchFieldException e) {  e.printStackTrace();  } catch (SecurityException e) {  e.printStackTrace();  } catch (IllegalArgumentException e) {  e.printStackTrace();  } catch (IllegalAccessException e) {  e.printStackTrace();  }  }  } |

### 设置私有属性

|  |
| --- |
| package com.oracle.test;  import java.lang.reflect.Field;  public class Test5 {  /\*\*  \* @param args  \* 类对象：代表类本身的对象,可以利用这个类对象来获取类本身的相关的信息比如这个类的属性有哪些，有哪些方法，  \* 有哪些构造方法，有哪些方法是公有的(public)和私有的（private）  \*/  public static void main(String[] args) {  // 使用反射的方式给对象的私有的属性赋值  // 1.创建学生对象  Student stu = new Student();  // 原来的属性赋值方式(不允许 因为age private)  // stu.age = 18;  // 使用反射的方式给对象的私有属性赋值  Class c = stu.getClass();  try {  Field f = c.getDeclaredField("age");  f.setAccessible(true);  f.set(stu, 18);  } catch (NoSuchFieldException e) {  e.printStackTrace();  } catch (SecurityException e) {  e.printStackTrace();  } catch (IllegalArgumentException e) {  e.printStackTrace();  } catch (IllegalAccessException e) {  e.printStackTrace();  }  }  } |

## Method类

|  |
| --- |
| package com.oracle.test;  import java.lang.reflect.InvocationTargetException;  import java.lang.reflect.Method;  public class Test6 {  /\*\*  \* @param args  \* 类对象：代表类本身的对象,可以利用这个类对象来获取类本身的相关的信息比如这个类的属性有哪些，有哪些方法，  \* 有哪些构造方法，有哪些方法是公有的(public)和私有的（private）  \*/  public static void main(String[] args) {  // 使用反射的方式给调用对象的方法  Student stu = new Student();  // 原来调用对象的方法的方式  // stu.study(10, 10.5);  // 使用反射的方式给调用对象的方法  Class c = stu.getClass();  try {  Method m = c.getDeclaredMethod("study", new Class[] { int.class,  double.class });  m.invoke(stu, new Object[] { 10, 10.5 });  } catch (NoSuchMethodException e) {  e.printStackTrace();  } catch (SecurityException e) {  e.printStackTrace();  } catch (IllegalAccessException e) {  e.printStackTrace();  } catch (IllegalArgumentException e) {  e.printStackTrace();  } catch (InvocationTargetException e) {  e.printStackTrace();  }  }  } |

## Constructor类

|  |
| --- |
| package com.oracle.test;  import java.lang.reflect.Constructor;  import java.lang.reflect.InvocationTargetException;  public class Test7 {  /\*\*  \* @param args  \* 类对象：代表类本身的对象,可以利用这个类对象来获取类本身的相关的信息比如这个类的属性有哪些，有哪些方法，  \* 有哪些构造方法，有哪些方法是公有的(public)和私有的（private）  \*/  public static void main(String[] args) {  // 使用反射的方式实现利用构造方法初始化属性  Student stu = new Student();  // Student stu = new Student("zzz",18);  // 获取类对象  Class c = stu.getClass();  try {  Constructor con = c.getDeclaredConstructor(new Class[] {  String.class, int.class });  Student s = (Student) con.newInstance(new Object[] { "zzz", 18 });  System.out.println(s.getAge() + " " + s.name);  } catch (NoSuchMethodException e) {  e.printStackTrace();  } catch (SecurityException e) {  e.printStackTrace();  } catch (InstantiationException e) {  e.printStackTrace();  } catch (IllegalAccessException e) {  e.printStackTrace();  } catch (IllegalArgumentException e) {  e.printStackTrace();  } catch (InvocationTargetException e) {  e.printStackTrace();  }  }  } |

# 反射的作用